Debugimine sardsüsteemides

Erkki Moorits
Cybernetica AS, Navigatsioonisüsteemide osakond

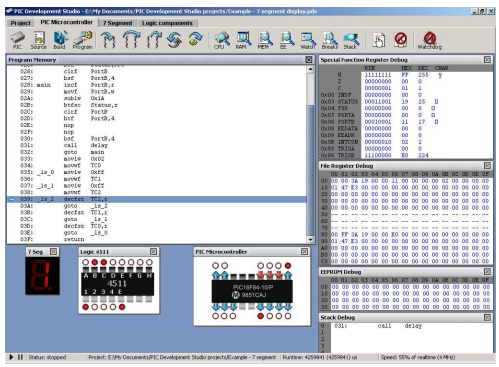
Sardsüsteemides kasutatavad debugimise riistvara ja meetodid

- Simulaatorid
- JTAG
- Emulaatorid
- API emulaatorid
- Debug terminaalid
- Riistvaraliste signaalide järgi
- Profileerimine

Simulaatorid vs. Emulaatorid

Järgnevalt on simulaatorite all mõeldud neid programme mis jäljendavad mittetäielikult protsessori või operatsioonisüsteemi tööd, näiteks ei võimalda reaalaja ülesandeid simuleerida või on muud moodi puudulikud. Seevastu emulaatorite all on üldiselt mõeldud riistvara mis jäljendab täpselt uuritavat süsteemi.

Simulaatorid



Simulaatorid

- Plussid
  - Kõige odavam viis riistvaralähedase programmi tööd kontrollida
  - Praktiliselt igale väiksemale kontrollerile olemas
  - Võimalik tarkvara ja riistvara koos simuleerida
- Miinused
  - Võib tekkida probleeme isegi simuleeritava riistvara enda eduka simuleerimisega
  - Võimaldab jälgida vaid neid programme, mis ei suhtle üldse või suhtlevad väga vähe ja aeglase välise riistvaraga
  - Aeglane

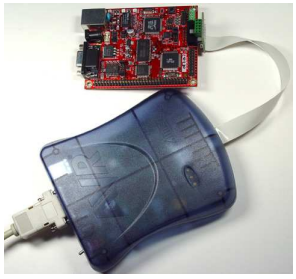
Tüüpiline simulatsiooni käik

- Kompileeritakse töötav programm, mis laetakse simulaatorisse, simulaator võib olla ka gdb osa. Selle sammu teevad ka kõik IDE'd (Integrated Development Environment)
- Kui simulatsiooni tulemus sõltub välistest signaalidest, siis tuleb eelnevalt kirjeldada kõiki väliseid signaale
- Vajadusel määrata simulatsiooni parameetrid ja breakpoint'id
- Käivitatakse simulatsioon

Kus ja kuna kasutada simulaatorit

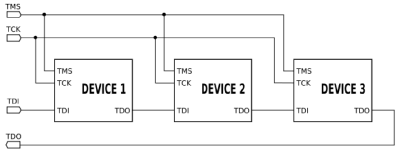
- Kui pole muid riistvaralisi debugimise meetodeid
- Kui simuleerimine on tunduvalt ohutum või odavam kui reaalselt seadme tarkvara testimine
- Kui on vajadus tarkvara testida selliste sissendparameetrite juures mida muidu ei ole võimalik saada

JTAG (Joint Test Action Group, IEEE 1149.1)



JTAG

- Algselt loodud tootmisel seadmete kontrollimiseks, kuid sobib ka programmide debugimiseks ja tarkvara laadimiseks
- On nõutud ainult 4 signaali (TDI, TDO, TCK, TMS), viies on valikuline (TRST), ehk tegemist on seriaal liidesega



JTAG

- Miks JTAG on nii laialt levinud?
  - Üks odavamaid riistvaralisi meetodeid millega programmi tööd jälgida
  - Praktiliselt kõik protsessorid/mikrokontrollerid võimaldavad JTAG'iga debugimist
- Probleemid JTAG'iga debugimisel
  - Riistvara millel JTAG'i kasutatakse peab täpselt jälgima JTAG'i käske
  - Keerukas suuri programme kontrollida
  - Kõrge protsessori takti juures kasutamine raskendatud
  - Ei pruugi võimaldada protsessori sammude ajalugu talletada

Mida JTAG võimaldab

- Tarkvara laadimiseks kontrollerisse, seda suudavad kõik JTAG'i moodulid teha
- Kerneli käivitamisel toimuva või mõne muu riistvara lähedase programmi debugimiseks
  - Võimalik seada programmi hardware ja software breakpointe.
- Tootmisel mikroskeemide ja nende ühenduste kontrollimist

Hardware vs. Software breakpoint

- Software breakpoint
  - *Software breakpoint* on protsessori peatamise instruktsioon mis on pannakse debugimise ajal **ajutiselt** programm mällu. NB! Seadmetel millel on flash mälu lühendab selline tegevus flashi eluiga.
- Hardware breakpoint
  - Ülejäänud protsessoritst sõltumatu elektroonika, mis jälgib protsessori tööd ja etteantud tingumusel peatab protsessori töö.

JTAG'i käskude mittejälgimine

Mõningatel kontrolleritel ei pane JTAG kontrolleri asünkroon taimereid peale *breakpointi* saabumist koheselt pausile. See põhjustab omakorda asünkroon taimeril uue katkestuse ja ühtlasi muudab JTAG'iga debugimise võimatuks.

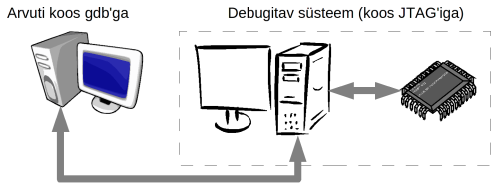
JTAG ja MMU (Memory Management Unit)

- MMU tegeleb arvutil ülesanneteks on aadresside transleerimine ja mälualade kaitse
- JTAG peab aru saama MMU omapäradest
  - MMU rakendamisel transleeritakse kõik füüsilised aadressid virtuaalseteks, mis kõik võivad alata samast aadressist – JTAG ei tohi “segadusse sattuda” nähes mitmes kohas sama aadressi (*run mode support*)

JTAG'i kasutamine

- JTAG on kasutatav ainult koos vastava debugeriga, enamusjuhtudel on selleks *gdb* (GNU Debugger)
- JTAG'i kasutamine on väga sarnane simulatsiooniga, *gdb* puhul on ainukeseks erinevuseks on debugeri teine back-end

Tüüpiline JTAG'i ühendus



- Debugitavat süsteemi ja debugerit ühendav link võib olla seriaalliides, IEEE 1394 (FireWire), ethernet, jne
- Debugitav süsteem võib olla simulaator või emulaator

JTAG'i kasutuskohad

- Arenduses programmi töö debugimiseks
- Tootmises seadme elektroonika korrasoleku kontrolliks

Emulaatorid

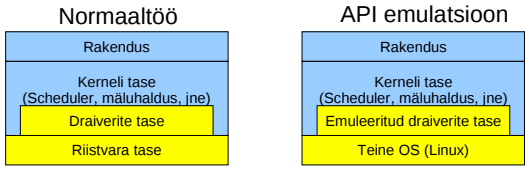
- Riistvaraliste emulaatorite eelised
  - Käituvad täpselt nagu emuleeritav riistvara
  - Võimalik üpriski pikka protsessori sammude ajalugu salvestada
- Riistvaraliste emulaatorite puudused
  - Vähemalt suurusjärk kallim kui JTAG'id
  - Uutele kiiretele protsessoritele/mikrokontrolleritele ei pruugi neid olla, ega ka tulla
  - Ei pruugi samasuguste elektriliste parameetritega olla nagu on emuleeritav seade ise
  - Rakendusprogramme keerukas debugida

Emulaatorite kasutuskohad

- Kernelite ja muu riistvaralähedase koodi debugimiseks nendes kohtades kus jääb JTAG'i võimalustest puudu

API emulaatorid

API emulaator on operatsioonisüteemi teekide ja draiverite kogu, mis on kompileeritud nii, et ei suhtle otse riistvaraga vaid suhtleb läbi teise OS'i.



API emulaatorid

- API emulaatorite kasutuskohad
  - Rakendusprogrammide kontroll
  - Kerneli mitte riistvaraga suhtlevate osade kontroll
- Puudused
  - Ei võimalda riistvaralähedaselt programme emuleerida
  - Väga väheseid kerneleid annab API emulatsiooniga kompileerida
  - Töökiirus sõltub peremees operatsioonisüsteemist

Kus kasutada API emulaatoreid

- Programmi kontseptsiooni või idee kiireks kontrolliks ja realiseeritavust kasutatava operatsioonisüsteemiga
- Rakendusprogrammide debugimiseks
- Kerneli poolt pakutavate funktsioonide kontrolliks

Debug terminaal

Debug terminaaliks kutsutaks ühte väljundporti, mis on reeglina seriaalport, kuhu kontroller saadab oma vea või olekuteateid. Unixi laadsetel operatsioonisüsteemidel (Linux, *BSD) on samade ülesannetega väljundiks süsteemi konsool (*system console*). Debug terminaali ülesandeid võib mõnikord täita ka teenindusprogramm.

Debug terminaalid

- Miks kasutada debug terminale
  - Hinna ja efektiivsuse suhtelt kõige parem
  - Võimaldab praktiliselt terve kerneli tööd jälgida
- Puudused
  - Mõningatel juhtudel võib olulisel määral mõjutada programmi tööd
  - Ajakriitilistes või piiratud mäluga kohtades ei ole võimalik kasutada
  - Riistvaral ei pruugi olla ühtegi vaba väljundporti (UART, USB, jne.)

Debug terminali kasutamine

```
#include <stdio.h>

#ifndef DEBUG_OUT
#define DEBUG_OUT stdout
#endif

#ifdef HAS_TRACE
#define TRACE fprintf (DEBUG_OUT, "%s @ %u\n", __FILE__, __LINE__)
#else
#define TRACE {}
#endif

int main (void)
{
    puts ("Hello world!");
    TRACE;
    puts ("Hello world! Again");
    return 0;
};

/*---- väljund ----*/
Hello world!
example-debugterm.c @ 16
Hello world! Again
```

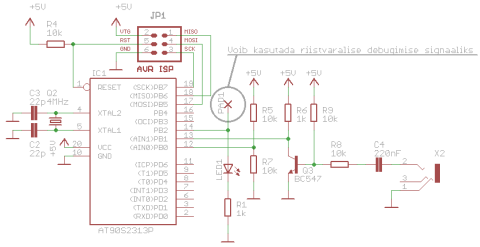

Mõned märkused debug terminalide kohta

- Debug terminali ei saa kasutada või on kasutamine raskendatud järgnevates kohtades:
  - Kerneli käivitamise alguses
  - Enamustes bootloaderites
  - Katkestustes
- Mikrokontrolleritel, seda eriti Harvardi arhitektuuriga seadmetel võib hooletu TRACE käskude lisamine võtta märkmisväärse osa mälust

Kus kasutada debug terminale

- Lõplikus tootes süsteemi seadistamiseks ja töö jälgimiseks
- Arenduse käigus debugimiseks

Riistvaralised signaalid



Riistvaralised signaalid

- Riistvaraliste signaalide eelised
  - Paljudel juhtudel ei maksa ühe signaali väljatoomine eriti palju
  - Enamustel süsteemidel on olemas mingi oleku indikaator või mõni vaba väljund mida saab vastavalt programmi olekule lülitada
  - Vastava aparatuuri olemas olul annab väga kiireid programmilõike jälgida, isegi schedulere
- Puudused
  - Riistvaraliste signaalidega annab kontrollida ainult lihtsate programmilõikude tööd
  - Programmi debugimine on väga aeganõudev
  - Keerukatel süsteemidel raske rakendada

Kus kasutada riisvaraliste signaalide järgi debugimist

- Väikestes või hobi projektides programmi töö kontrolliks
- Lõplikus tootes süsteemi osa üldise oleku näitamiseks, kas töötab või ei tööta
- Tootearendusel mõne väga ajakriitilise programmilõigu jälgimisel
- Piiratud võimalustega mikrokontrolleri programmi debugimisel (näiteks PIC10F200, mis on SOT23-6 korpuses)

Profileerimine

Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing. This information can show you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execute faster. It can also tell you which functions are being called more or less often than you expected. This may help you spot bugs that had otherwise been unnoticed.

(http://sourceware.org/binutils/docs-2.20/gprof/index.html)

Profileerimine sardsüsteemides

- 8 ja 16 bitistel mikrokontrolleritel ning isegi väiksematel ARM'idel ei ole võimalik kasutada enamlevinud profileerimise rakendusi, näiteks *gprof*'i
- Alternatiivsed variandid:
  - Profileerimiseks on võimalik kasutada simulaatoreid, mis programmi täites loendavad etteantud sündmusi
  - Võimalik kirjutada kerneli koodi sisse vastavaid profileerimise funktsioone ja pärast väljastada tulemus debug porti, või mujale väljundisse

Debugimis meetodite võrdlus

Meetod / Kasutuskoht	Kerneli debugimine	Rakendus debugimine	Mõju seadme tööle
Simulaator	Enamusjaolt võimalik	Aega nõudev	Puudub
JTAG	Enamusjaolt võimalik	Raskendatud või ebaefektiivne	Vähestel juhtudel võib mõjutada
Emulaator	Võimalik	Raskendatud või ebaefektiivne	Praktiliselt olematu
API emulaator	Osaliselt võimalik	Võimalik	Puudub
Debug terminaal	Enamusjaolt võimalik	Võimalik	Oleneb kasutuskohast, mõnikord võib olulisel määral mõjutada
Riistvaralised signaalid	Võimalik, kuid võib olla ebaefektiivne	Praktiliselt võimatu või ebaefektiivne	Minimaalne
Profileerimine	Võimalik, kuid võib põhjustada arusaamatuid viiteid	Võimalik	Oleneb kasutuskohast, mõnikord võib olulisel määral mõjutada

Kokkuvõte

- Debugimis meetodi valik sõltub väga olulisel määral süsteemist ja rakendusese mida debugida soovitakse
- Praktiliselt kõik debugimise meetodeid võivad mingil määral mõjutada süsteemi toimimist
- Arenduses sobivad debugimiseks kõige paremini JTAG ja debug terminaal
- Tootmises sõltub debugimise meetod seadme eripärast, võivad sobida nii lihtne riistvaraline signaal, JTAG ja debug terminaal
- Väljastatud tootes sobivad kõige paremini seadme kontrolliks lihtne riistvaraline signaal ja debug terminaal